

Assignment 4

CS 350-Spring 2005

v. 3/23/05

Assignment Overview

1. Test several versions of ARSP for correctness. The code to be tested is provided in object format.
2. Prepare sufficient test data to confirm the correctness of ARSP.
3. Prepare a test report describing the results of your unit testing.
4. Share your test data with other team members.
5. Test ARSP with test data from other team members.
6. Prepare a test report describing the results of this testing.

Test Driver Requirements

The basic outline of the test driver for this assignment might be:

```
// Initial comments
...

// Decl of global vars
...

// Decl of external functions (here, the function to be tested)
...

int main( );
{
// decl of statistics vars
int TestCaseCt = 0,
    TotalMatchCt = 0,
    TotalMismatchCt = 0;
...

// Read test data until eof
// Precondition: values are in the order specified; no error handling is
// included; if the data file is incorrectly ordered, or too few or
// too many values are provided, the program may not behave sanely.
cin >> so.ar_altitude[0] );
while ( cin )
{
// Initialize/adjust statistics for this iteration.
TestCaseCt++;
MatchCt = 0;
MismatchCt = 0;

// Initialize remaining globals.
...

// Call ftn being tested.
arsp();

// Compare and report matches of actual and expected values
if ( actual value <> expected value {
    count mismatch;
    display mismatched values;
else
    count match;
```

```

// Print statistics for this test case.
cout << "Results from test case " << TestCaseCt << endl;

    ...
    cout << "    Number matches:    " << MatchCt << endl;
    cout << "    Number mismatches:" << MismatchCt << endl;

// Set up for next test case.  If no more data in file, this
// will cause the condition in the while loop to be false.
    GetNextNum( ex.ae_cmd[0] );
}

// Print summary statistics
cout << "Summary Statistics" << endl;
cout << "    Number of test cases run: " << TestCaseCt << endl;
cout << "    Total match count for all test cases:    "
    << TotalMatchCt << endl;
cout << "    Total mismatch count for all test cases: "
    << "TotalMismatchCt" << endl;
}

```

Note how the while loop is set up in the pseudocode above so that the program will keep reading test cases until end of file. Also note that no specific error handling is required even if data files are improperly formatted. If input data do not conform to the required format, then your program need not function correctly. This is important because it is possible that some of the test data you get from team members will not be properly formatted; don't worry about coding to handle this possibility.

Test Cases and Testing Objectives

Beginning programmers are usually not skilled testers. For many projects, as much time, effort and creativity goes into testing as coding. Creating effective test data is a part of the programming task; in many software development organizations, you are responsible for unit testing the code you write using data you have created.

A frequently asked question with this type of assignment is: "how many test cases should I create?" Unfortunately, answering question this is part of this assignment. For an arbitrary program, a reasonable answer usually requires careful analysis of the program requirements and an understanding of the consequences of code failure. The number of appropriate test cases can vary significantly even for similarly sized modules.

To help, here are some testing goals you might consider. You should make sure that you have at least one test case to check each goal. It is suggested that, since it is common for code requirements to change both during development and after release, each test case should check only one goal if feasible.

One way of deciding if a given set of test cases is adequate is 1) to make sure every requirement is checked by at least one test case, and 2) that every test case is justified by a particular requirement. Preparing a list of testing objectives is a way of planning and documenting that this has occurred.

Recall that additional requirements are found in the front of the spec. You will likely need additional test cases for them. In addition, creating effective test cases generally requires that you imagine the types of mistakes that a programmer might make and test for those.

Test Data Formatting Requirements

Since you are to share test data with team members, the test files must be a format that all team member's test drivers can read, your team will need to agree on a common format. I suggest that

the data be read into variables in alphabetical order by variable name. Since some the variables are arrays, you must also agree on the order of values for each array.

Each test file must contain both input values (to be used by the arsp code being tested) and the correct output values that the arsp code should produce. Thus the data file should contain values for `ar_altitude` as input and values for `ar_altitude` as output; likewise for `ar_status` and `k_alt`.

Hints on Naming Files and Running Tests

In order to facilitate reuse of individual test cases, each must be in a separate file. You should use a simple pattern for naming each of these individual test case files, such as “TestCase01,” “TestCase02,” ..., “TestCase35.” Then if you name your test driver “prog4driver” and have all data files in a single directory (which you should), you could run all test cases with the single command:

```
cat TestCase* | prog4driver > TestReport1
```

Required Output

Your test driver must produce statistics on the number of values read, the number of matches, the number of mismatches, and the number of testcases run; see the code outline above. Your code should also display mismatch values and the name of the variable.

Location of ARSP Versions To Be Tested

You should test the 5 versions of ARSP located in `/home/cs350/term.projects/test.modules/arsp`. You should not copy these into your home directory but link to them in their current location.

Contents of Test Reports

Produce test reports for each of the testing activities that summarize the results of that testing activity including: 1) your code with your own data (one report), and 2) your code with other team member data (another report).

Each test report should contain:

1. Your name.
2. The version of `arsp` tested
3. The date or dates the testing was conducted.
4. The name of the file generated by your tester.
5. Then for each test case:
 - a. Identify the test case filename and test number.
 - b. State the test case objective for this test case. This is usually a one sentence description such as “Determine if the variable `ar_altitude` is rotated correctly.”
 - c. Describe results from that test case, for example:
 - i. all program output matched the expected values from the test data file.
 - ii. code did not link correctly; unable to produce an executable.
 - iii. format of provided test data file incorrect.
 - iv. test data file has incorrect expected outputs. (State which variables you believe to be wrong. Explain why.)
 - v. run-time error, no output produced.
 - vi. both the actual and expected values for each mismatch, including the variable name
 - d. Include a count of the number of data values matches and the number of mismatches.
6. An overall summary of testing results for testing, including the total number of matches and mismatches.

A reader of your test report must be able to determine which test data file corresponds to which test case. **Warning:** Lots of unexpected things can happen in testing, the purpose test report is to summarize whatever happened; you have a flexible format to allow this. **Remember, not all code to be tested actually executes. If so, your test report should state this—any other problems you encountered with the code and data you are to test.**

Deliverables

You must submit each of the following using the ODU submit command on UNIX:

1. The source code of your driver along with a Makefile.
2. Your test data. This should be placed in one directory and that directory submitted. You must also include a file that provides the test objective for each test case.
3. Your test report from testing your code with your data. Include all files generated by your tester that are reported on in the test report.

Different parts of this assignment are due on different dates. See the due dates file on the class web site for this assignment.

Grading

- | | |
|--|-----------|
| 1. Properly structured and commented code: Note that your assignment will be returned ungraded if this is not so. | 0 points |
| 2. Correctness/completeness of your test driver: | 50 points |
| 3. Effectiveness of your test data: | 50 points |